PROJECT REPORT
MAGGOT THERAPIST

UE18CS152
PROBLEM SOLVING WITH C LABORATORY

TEAM:
B K Karthik – PES2201800185
Joe Rishon Manoj - PES2201800340
Joshua Phillips – PES2201800333
Manav Agarwal – PES2201800025

SOURCE CODE:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<unistd.h>
/*
DEFINING CUSTOM MACROS FOR KEEPING A TRACK OF THE PROCESS
*/
#define _DONE_ "\ndone\n"
#define _NOT_DONE_ "\nnot done\n"

/*
GLOBAL VARIABLE TO KEEP A TRACK OF THE TOTAL NUMBER OF ERRORS
*/
int NUMBER;

/*
struct error IS THE USER DEFINED DATA TYPE OF A TYPICAL SYNTAX ERROR
*/
struct error{
int lineNo;
char replace[500];
};

/*
ALIAS FOR struct error ERROR
*/
typedef struct error ERROR;

/*
USER DEFINED FUNCTION IN GLOBAL SCOPE PRINTS "done" IF THE RETURN VALUE OF THE FUNCTION WAS SUCCESSFULL
PRINTS "not done" IF THE RETURN VALUE OF A FUNCTION WAS NOT 0
*/
```

```c
extern void print(int argc){
if(argc==0){
fprintf(stdout,"%s",_DONE_);
//PRINTING TO STANDARD OUTPUT STREAM
}
else if(argc==-1){
fprintf(stdout,"%s",_NOT_DONE_);
//PRINTING TO STANDARD OUTPUT STREAM
}
}

/*
USER DEFINED FUNCTION TO CLEAR ALL CLUTTER FROM THE STANDARD
INPUT,OUTPUT,ERROR STREAMS
*/

extern int start(void){
fflush(stdin);
fflush(stdout);
fflush(stderr);
//EXPLICIT FLUSH
fprintf(stdout,"%s","\nSTART\n");
fprintf(stdout,"%s","\nInitialisation...\n");
fprintf(stdout,"%s","\nPlease make sure there are no other significant background
processes running\n");
fprintf(stdout,"%s","\nPlease make sure the file to be worked on is named \"source.c\"\n");
//PRINTING TO OUTPUT STREAM
return 0;
//SUCCESS
//NO CHANCES OF FAILURES UNLESS THE STANDARD OUTPUT STREAM IS INACCESSIBLE, IN
WHICH CASE, THIS STATEMENT WOULD NEVER BE REACHED
}

/*
USER DEFINED FUNCTION TO REDIRECT AND PROCESS ERRORS
*/
extern int run(void){
fprintf(stdout,"%s","\nCOMPILATION...\n");
//PRINTING TO STANDARD OUTPUT STREAM
system(" gcc -w source.c 1>/dev/null 2>maggot.txt");
//CREATION OF SUBSHELL AND EXECUTING GCC WITHOUT LOGGING AND REDIRECTING
ERRORS TO A FILE AND DISCARDING ALL OUTPUT
fprintf(stdout,"%s","\nPROCESSING...\n");
//PRINTING TO THE STANDARD OUTPUT STREAM
system(" python3 process.py 1>error.txt 2>/dev/null");
//CREATION OF SUBSHELL AND EXECUTING GCC WITHOUT LOGGING AND REDIRECTING
OUTPUT TO A FILE AND DISCARDING ALL ERRORS
return 0;
}

/*
```

```
USER DEFINED FUNCTION TO EXTRACT BASIC PARAMS FROM THE PROCESSED ERROR FILE
*/
extern ERROR* getErrors(void){
//FILE POINTER IN READ MODE
FILE* fp=fopen("error.txt","r");
//ARRAY OF CHARECTERS USED TO READ LINES OUT OF THE FILE
char reader[500];
fgets(reader,500,fp);
fclose(fp);
//FILE CLOSED
//ASSIGNMENT TO GLOBAL VARIABLE
NUMBER=atoi(reader);
//DYNAMIC MEMORY ALLOCATION TO ACCOMODATE ALL PROCESSED ERRORS
ERROR* Errors=(ERROR*)malloc(NUMBER*sizeof(ERROR));
return Errors;
}
/*
USER DEFINED FUNCTION TO EXTRACT SPECIFIC PARAMS FROM THE PROCESSED ERROR
FILES
*/

extern int readErrors(ERROR* Errors){
//FILE POINTER
FILE* fp=fopen("error.txt","r");
//ARRAY OF CHARECTERS USED TO READ LINES OUT OF THE FILE
char reader[500];

fgets(reader,500,fp);
for(int i=0;i<NUMBER;i++){
fgets(reader,500,fp);
Errors[i].lineNo=atoi(reader);
fgets(reader,500,fp);
fgets(reader,500,fp);
strcpy(Errors[i].replace,reader);
}
return 0;
}

/*
USER DEFINED FUNCTION TO CORRECT THE ERRORS TAKING A POINTER TO THE
STRUCTURE ERROR AND RETURNING SUCCESS OR FAILURES BY MEANS OF AN INTEGER
*/
extern int replace(ERROR* Error){
//FILE POINTERS
FILE *fileptr1, *fileptr2;
const char filechar[9]="source.c";
char c;
//TEMPORARY COUNTING VARIABLE
int temp=1;
//printf("Enter file name: ");//change
//scanf("%s", filechar);
```

```
fileptr1 = fopen(filechar, "r");
c = getc(fileptr1);
//print the contents of file .
while (c != EOF)
{
printf("%c", c);
c = getc(fileptr1);
}
printf("\nAFTER MODIFICATION\nPLEASE COPY THE CODE ABOVE IF THIS MODIFICATION
WAS UNNECESSARY\n");
//take fileptr1 to start point.
rewind(fileptr1);
//open replica.c in write mode
fileptr2 = fopen("replica.c", "w");
//OPENING A NEW FILE IN WRITE MODE TO PREVENT MULTIPLE FILE POINTER MOVEMENTS
THAT COULD SLOW THE PROCESS DOWN
//AND KEEPING THE OPTION OF ABORTING THE PROCESS AND NOT LOSING ANY VALUABLE
INFORMATION OPEN
c = getc(fileptr1);
while (c != EOF){
//FINITE LOOP
if (c == '\n'){
temp++;
}
//till the line to be deleted comes,copy the content to other
if (temp != Error->lineNo){
putc(c, fileptr2);
}
else{
while ((c = getc(fileptr1)) != '\n'){
}
//read and skip the line ask for new text
//flush the input stream
fflush(stdin);
//TEMPORARY VARIABLE FOR COUNTING
int var=0;
for(var=0;var<strlen(Error->replace);var++){
putc(Error->replace[var],fileptr2);
}
fputs("\n", fileptr2);
temp++;
}
//continue this till EOF is encountered
c = getc(fileptr1);
}
fclose(fileptr1);
//FILE CLOSED
fclose(fileptr2);
//FILE CLOSED
remove(filechar);
//FILE DELETED
```

```
rename("replica.c", filechar);
//FILE RENAMED, SO THAT EVERYTHING LOOKS LIKE BEFORE
fileptr1 = fopen(filechar, "r");
//reads the character from file
c = getc(fileptr1);
//until last character of file is encountered
while (c != EOF)
{
printf("%c", c);
//all characters are printed
c = getc(fileptr1);
//READING EACH INDIVISUAL CHARECTER
}
fclose(fileptr1);
//FILE CLOSED
}
/*
__MAIN__
*/
int main(int argc,char** argv){
print(start());
//PRINTS DONE
print(run());
//PRINTS DONE]

ERROR* Errors=getErrors();
//NO OUTPUT
print(readErrors(Errors));
//PRINTS DONE
for(int i=0;i<NUMBER;i++){
fprintf(stdout,"\nError %d\n",i+1);
print(replace(&Errors[i]));
//PRINTS DONE
}
fprintf(stdout,"\nDEALLOCATING ALL MEMORY\n");
//PRINTING TO THE STANDARD OUTPUT STREAM
fprintf(stdout,"\nSTOP\n\n\n");
//EXIT

return 0;
}


import string
def work(mstr):
global fout
mstr=(mstr[9:])
for i in mstr.split(":")[:2]:
print(i)
for i in range(len(mstr)):
if(ord(mstr[i])==8216 and ord(mstr[i+2])==8217):
```

```
print(mstr[i+1],end="")
def func(mlist):
global fout
for i in range(len(mlist)-1):
work((str(mlist[i]))[:-1]) #this is the second function call
print((str(mlist[i+1]))[:-1])
def main(f,fout):
l=f.readlines(0)
for i in l:
if ("In function") in i or ("^~") in i or ("^") in i or ("note") in i or ("warning") in i:
l.remove(i)
for i in l:
if ("In function") in i or ("^~") in i or ("^") in i or ("note") in i or ("warning") in i:
l.remove(i)
print(len(l)//2) #this displays the total number of errors, because each error contributes to
2 list elements, 1 is the deials of the error and the other is the replace line itself
for i in range(0,len(l)-1,2):
func([l[i],l[i+1]]) #here is the first function call
fout.close()
f=open("maggot.txt","r")
fout=open("error.txt","w")
main(f,fout)
```

FILE USED FOR TESTING

BEFORE PROCESSING

```
#include<stdio.h>
int main(){
printf("line 1");
printf("line 2 with missing ;")
printf("line three and last line");

}
```

AFTER PROCESSING

```
#include<stdio.h>
int main(){
printf("line 1");
printf("line 2 with missing ;"); printf("line three and last line");


}
```

TERMINAL WINDOW

```
call-me-bk@BK:~/Desktop/Cproject/maggots$ ./a.out

START
```

```
Initialisation...

Please make sure there are no other significant background processes running

Please make sure the file to be worked on is named "source.c"

done

COMPILATION...

PROCESSING...

done

done

Error 1
#include<stdio.h>
int main(){
   printf("line 1");
   printf("line 2 with missing ;")
   printf("line three and last line");

}
AFTER MODIFICATION
PLEASE COPY THE CODE ABOVE IF THIS MODIFICATION WAS UNNECESSARY
#include<stdio.h>
int main(){
   printf("line 1");
   printf("line 2 with missing ;");    printf("line three and last line");


}
done

DEALLOCATING ALL MEMORY

STOP


call-me-bk@BK:~/Desktop/Cproject/maggots$
```